Data science and Economics, Department of Economics, Management, and Quantitative Methods

UNIVERSITÀ DEGLI STUDI DI MILANO

IMAGE CLASSIFICATION WITH NEURAL NETWORK

Glasses or no Glasses

Name: *Natasha Amjad*

Mat° *939736*

“I/We declare that this material, which I/We now submit for assessment, is entirely my/our work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offenses in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study. “

1 Table of Contents

Table of Figures

1 Summary:

The goal of this project is to create a classifier that can distinguish between those who are wearing glasses and those who don't. Kaggle was utilized to obtain the dataset for this project. The dataset was created with the help of a Generative Adversarial Neural Network (GAN), and it consists of feature vectors of images of people wearing or not wearing glasses. These images are created by the GAN network using 512 number latent vectors. Both the latent vectors and the faces produced by those vectors are provided by Kaggle. Both were used to determine whether a person was wearing glasses.

In this study, we used a Neural Network to train for binary classification and attempted to answer the following questions:

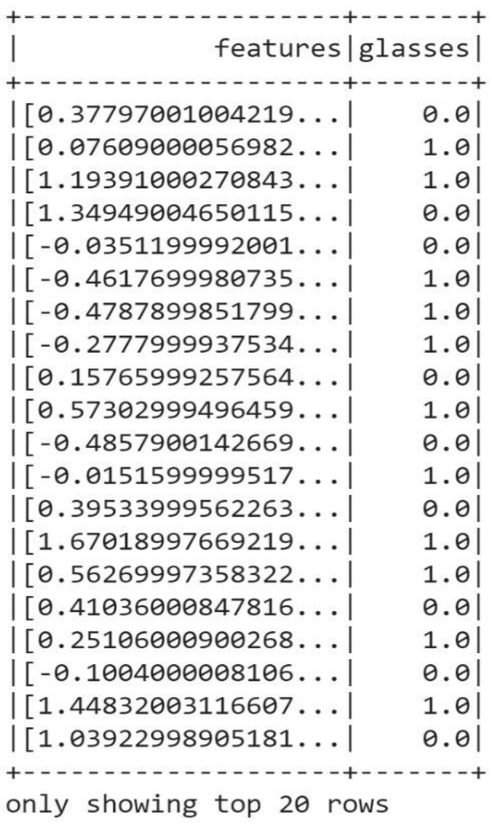
- How well does the model perform with the tuning of various hyperparameters?
- For image recognition, which popular architectures (AlexNet and LeNet-5) are more reliable?

This project was split into two sections. In both sections, we imported different datasets (i.e latent vectors and produced images dataset). The PySpark Apache framework was utilized in the first section, and we looked at model performance using hyperparameter tuning and model selection methods. In the second section, The Convolutional Neural Network was employed. We build a model based on CNN and try to predict image class by directly giving images as input and getting classified as an output. Moreover, two well-known architectures from the machine learning literature are used to check how the accuracy changes when these consolidated architectures are used, as well as to compare their performance and the potential problems caused by the networks' complexity. The architectures used in this part are specifically AlexNet and LeNet-5

Section I

2 Dataset and Pre-processing:

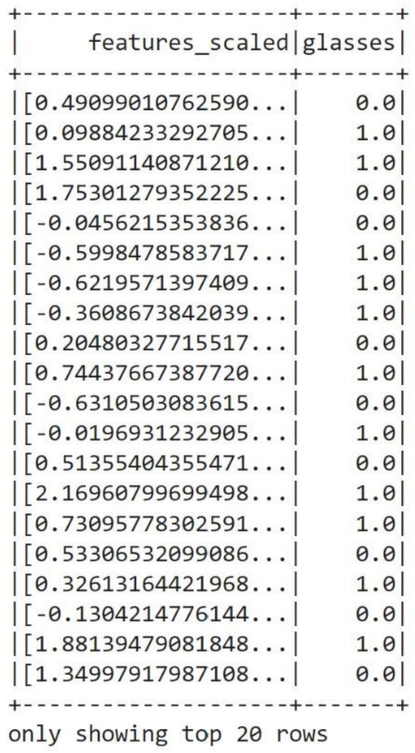
The GAN dataset is taken from Kaggle online community. The GAN CSV file is used as a training dataset. The training dataset consists of 4500 rows where each row represents a training image with 514 columns. In the last column, we have the feature we would like to predict. We stored the dataset in a PySpark data frame. We used feature pre-processing to convert double values to floats. In our dataset, we looked for NULL and duplicate values. After cleaning our dataset, we utilize a Vector assembler. Which is a transformer that creates a single vector column from a list of columns. It is used to train ML models like logistic regression by merging raw features and features generated by different feature transformers into a single feature vector. All integer types, Boolean types, and vector types are accepted by Vector Assembler as input column types.



```
+--------------------+-------+
|            features|glasses|
+--------------------+-------+
|[0.37797001004219...|    0.0|
|[0.07609000056982...|    1.0|
|[1.19391000270843...|    1.0|
|[1.34949004650115...|    0.0|
|[-0.0351199992001...|    0.0|
|[-0.4617699980735...|    1.0|
|[-0.4787899851799...|    1.0|
|[-0.2777999937534...|    1.0|
|[0.15765999257564...|    0.0|
|[0.57302999496459...|    1.0|
|[-0.4857900142669...|    0.0|
|[-0.0151599999517...|    1.0|
|[0.39533999562263...|    0.0|
|[1.67018997669219...|    1.0|
|[0.56269997358322...|    1.0|
|[0.41036000847816...|    0.0|
|[0.25106000900268...|    1.0|
|[-0.1004000008106...|    0.0|
|[1.44832003116607...|    1.0|
|[1.03922998905181...|    0.0|
+--------------------+-------+
only showing top 20 rows
```

The input columns' values will be concatenated into a vector in the desired order in each row. In our case, the input columns are from V1 to V512, and we concatenated them in one column named "Features" with an output column named "Glasses".

After assembling our dataset, StandardScaler is used to perform Feature Scaling. In general, we utilize Standard Scalar to scale the magnitude of a feature within a given range. In general, the data we collect from the real world differs significantly, and this has a direct impact on the model's performance. As a result, scaling the data before processing it is always a good idea. StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. We call the *fit_transform()* function and pass it to our dataset to create a transformed version of our dataset.



```
+--------------------+-------+
|     features_scaled|glasses|
+--------------------+-------+
|[0.49099010762590...|    0.0|
|[0.09884233292705...|    1.0|
|[1.55091140871210...|    1.0|
|[1.75301279352225...|    0.0|
|[-0.0456215353836...|    0.0|
|[-0.5998478583717...|    1.0|
|[-0.6219571397409...|    1.0|
|[-0.3608673842039...|    1.0|
|[0.20480327715517...|    0.0|
|[0.74437667387720...|    1.0|
|[-0.6310503083615...|    0.0|
|[-0.0196931232905...|    1.0|
|[0.51355404355471...|    0.0|
|[2.16960799699498...|    1.0|
|[0.73095778302591...|    1.0|
|[0.53306532099086...|    0.0|
|[0.32613164421968...|    1.0|
|[-0.1304214776144...|    0.0|
|[1.88139479081848...|    1.0|
|[1.34997917987108...|    0.0|
+--------------------+-------+
only showing top 20 rows
```

Figure 2-1 Scaled feature

The dataset has been randomly split into train and test datasets with a size of 20% and 80%.

3 Model I

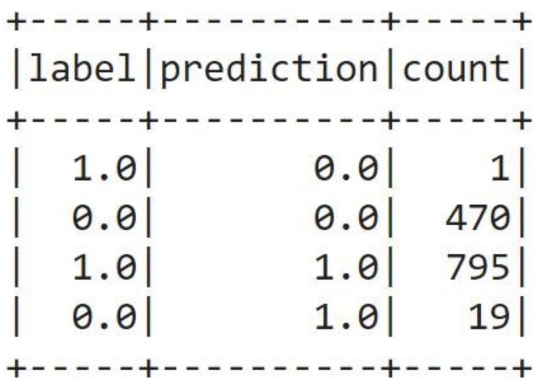
3.1 Hyperparameter tuning with Cross-validation

In PySpark we can use cross validator tool for selecting our model. These tools require an Estimator which we call an algorithm, a set of ParamMaps sometimes we call “parameter grid” to search over, and Evaluator to measure the model performance. CrossValidator split the input data into training and test datasets. For each pair, they iterate through the set of ParaMaps: They fit the Estimator with those parameters for each ParamMap, get the fitted Model, then use the Evaluator to evaluate the Model's performance. As a result, the best-performing model generated by a set of parameters was chosen in the end. Since we are dealing with Neural Networks, we utilized MultilayerPreceptronClassifier as an estimator and for evaluation binaryclassificationEvaluator.

Grid search is a technique for determining the best set of hyperparameters for a model. The ParamGridBuilder is used to build the parameter grid. We build this model bit complex adding a high number of neuron layers (512,256,128,64,2). We set three values for block size (64, 128, 256) and three for maxIter (50, 100, 150) in the parameter grid. The crossvalidator employs five-folds, yielding 45 distinct trained models. We fit the model and trained it which is very expensive.

3.2 Predictions and model evaluation

To evaluate the performance of our classifier we built a matrix. As we can see our model is performing well overall but still has a little high rate of False-positive value. The accuracy of our model is 98%. We also computed the F1 score (98%), which is harmonic to precision and recall. It takes both FN and FP values into account.



```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       0.0|    1|
|  0.0|       0.0|  470|
|  1.0|       1.0|  795|
|  0.0|       1.0|   19|
+-----+----------+-----+
```

Figure 3-1 Confusion Matrix

4 Model II

4.1 Hyperparameter tuning with train validation split

In Spark, we can also use train-validation split for hyperparameter tuning. In contrast to CrossValidator, Train-Validation Split only examines each combination of parameters once, rather than k-times. As a result, it is less expensive, but it will not generate as trustworthy results if the training dataset is not large enough. It splits the datasets into two parts using the trainRatio parameter. It could be like 80% of data used for training and 20% for validation. Then we finally fit the Estimator MLP using the best ParamMap and the entire dataset.

In this model, we use a less complex network. The layers we used here are (512,128,64,2) with blockSize (32,64,128) and maxIter (50,100,150). We fit our model and train it.

4.2 Predictions and model evaluation

With a less dense model, the accuracy drops, and even with the same F1 score, we get a 97 percent accuracy. Our model is performing overall well but we get an even higher false-positive score as compared to the previous model.

label	Prediction	count
1.0	0.0	0
0.0	0.0	460
1.0	1.0	796
0.0	1.0	29

Figure 4-1 Confusion Matrix

Section II

5 Dataset and pre-processing:

In the subject of image recognition, face recognition is a fascinating application. Face images with spectacles, on the other hand, are a hindrance. The purpose of this project's section is to divide people's photos into two categories: Glass and No-Glass are depending on whether they are wearing glasses in the image. However, glasses come in a wide range of shapes, designs, and appearances, making them difficult to choose. The data was obtained via the Kaggle online community. We have a single file that contains all the photographs of people wearing and not wearing glasses. Then there are separate image classes in each file. There is no need for pre-processing because the dataset has already been cleansed and is free of noise. The dataset is stored in a data frame and contains a total of 4920 photos, with 2769 images belonging to class glasses and 2151 to no-glasses. We split our dataset into two parts: a train set and a validation set, with an 80-20 percent split.

6 Model Training

We used all image datasets to generate a train and test dataset. ImageDataGenerator was used to create batches of images with real-time data augmentation. To assist generate more images of such properties, the ImageDataGenerator can take in a lot more inputs like shear range, zoom range, horizontal flip, and vertical flip. We try to plot a few images taken from our training dataset which are given below:

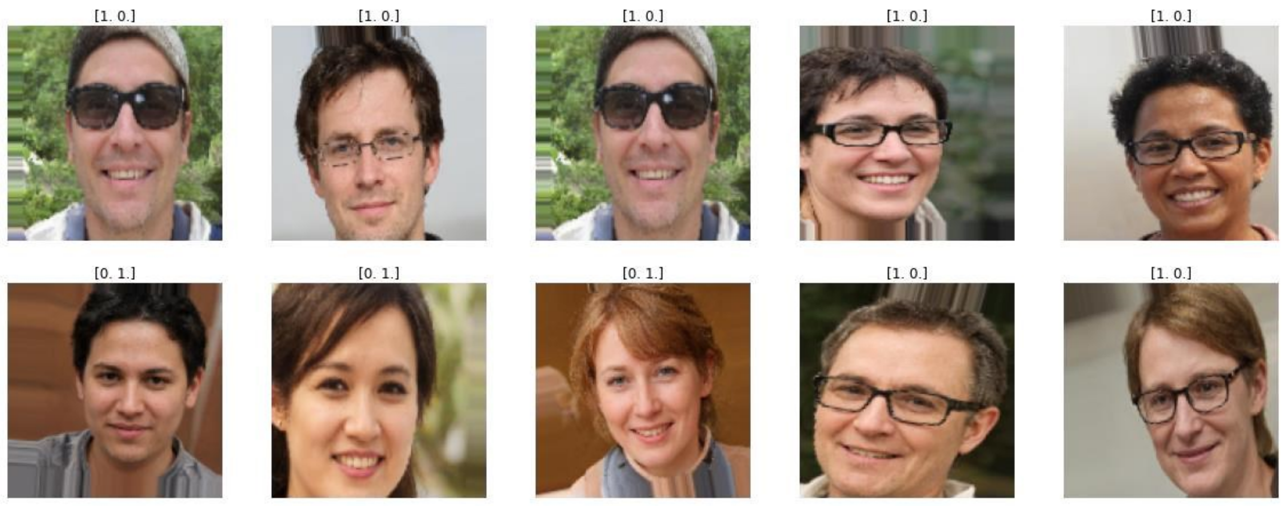
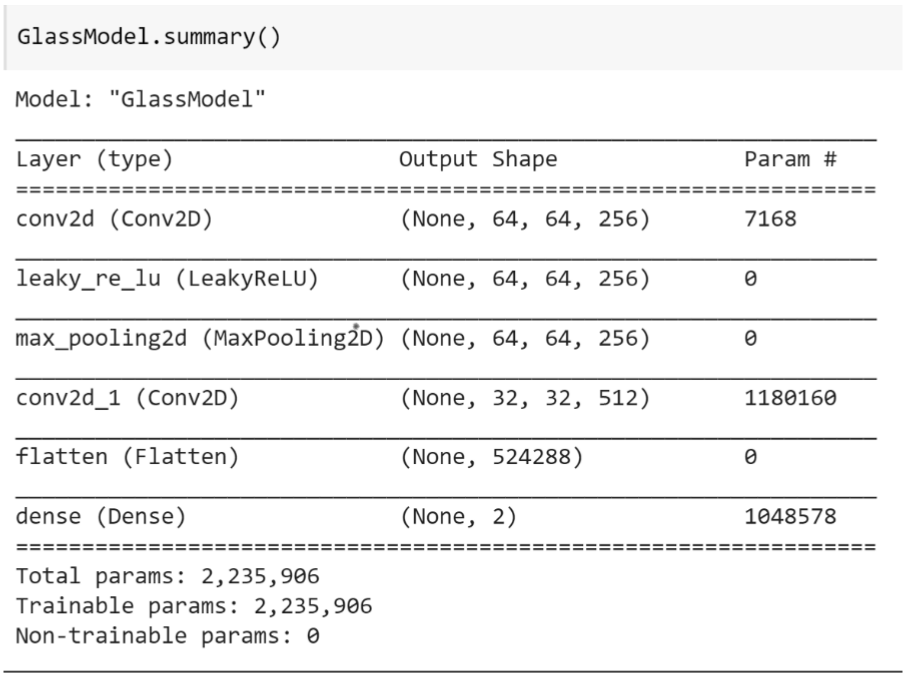


Figure 6-1 Training Image dataset

We build the convolutional Neural Network architecture which contains: 6 layers in total:

- 2 Conv2d
- a Leaky-reLU
- 1 Max_pooling2d
- 1 Flatten
- 1 Dense



```
GlassModel.summary()
```

```
Model: "GlassModel"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 256)       7168
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 64, 64, 256)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 256)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 512)       1180160
_________________________________________________________________
flatten (Flatten)            (None, 524288)            0
_________________________________________________________________
dense (Dense)                (None, 2)                 1048578
=================================================================
Total params: 2,235,906
Trainable params: 2,235,906
Non-trainable params: 0
_________________________________________________________________
```

We use several techniques while training our model. Overfitting the training dataset can come from using too many epochs, whereas using too few can result in an underfit model. First, we use early stopping, which is a technique that lets you specify an arbitrarily large number of training epochs and then stop training once the model's performance on a holdout validation dataset stops improving. For this purpose, we set the patience value to 10 means training will stop if there is no improvement in model performance. Secondly, we use learning-rate-reduction. When learning gets stationary, models often benefit from slowing the learning rate by a factor of 2-10. This callback observes a quantity and reduces the learning rate if no improvement is noticed after a 'patience=2’ number of epochs in our model. To use our model later we saved it as hdf5 by using the checkpointing technique. We train the model with 10 epochs and use Adam optimizer. The accuracy and Loss for the training and test dataset are shown in the graph below:

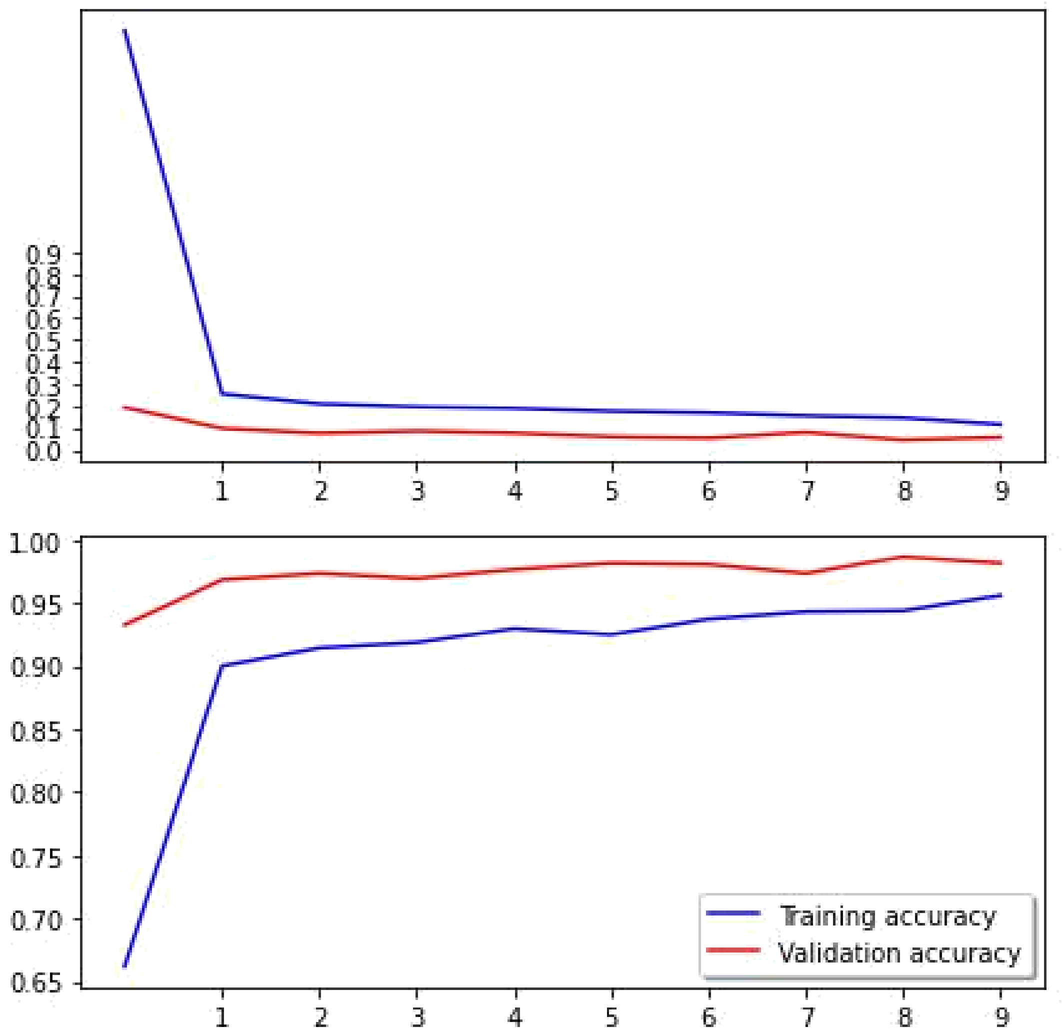


Figure 6-2 Accuracy and loss for training and validation dataset

7 Performance Evaluation

To evaluate the performance of our classifier we built a confusion matrix. Our model performed pretty well. It achieved an accuracy of 99 percent. As we can see our model is performing well overall but still has a little high rate of False-positive and False-negative values.

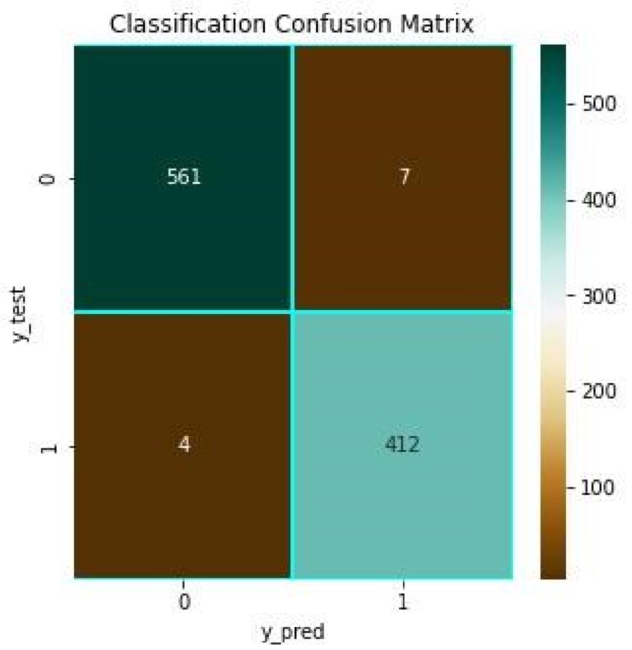
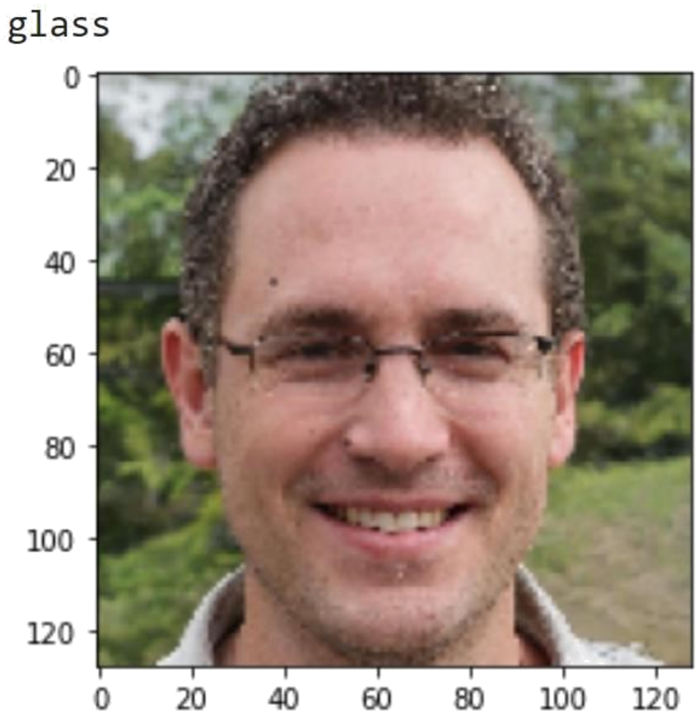


Figure 7-1: confusion matrix CNN Model

After the model was trained and evaluated, we saved it in an hdf5 file. This model is used to predict Image classes directly. We give an image as an input, and it shows the class it belongs to whether glasses or no-glasses. Our Predicted images are shown below:



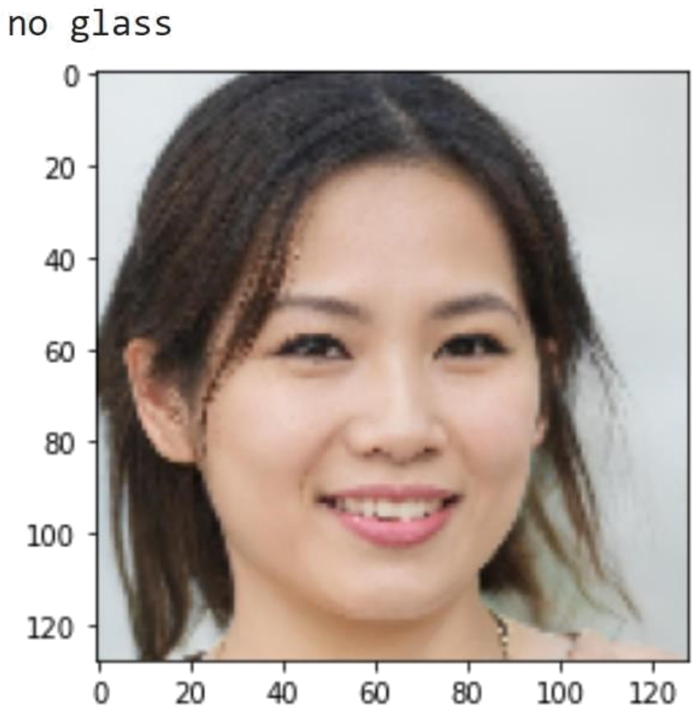


Figure 7-2 Predicted images by Model

8 AlexNet Architecture

AlexNet is an eight-layer network proposed by Alex Krizhevsky in his work, including five convolutional layers followed by three fully linked layers. The network is consisted of five layers, for each of its five convolutional layers, the network uses a kernel or filters with sizes of 11 x 11, 5 x 5, 3 x 3, 3 x 3, and 3 x 3. The network's remaining parameters can be fine-tuned based on training results. It uses ReLU as an activation function at all layers except at the output layer.

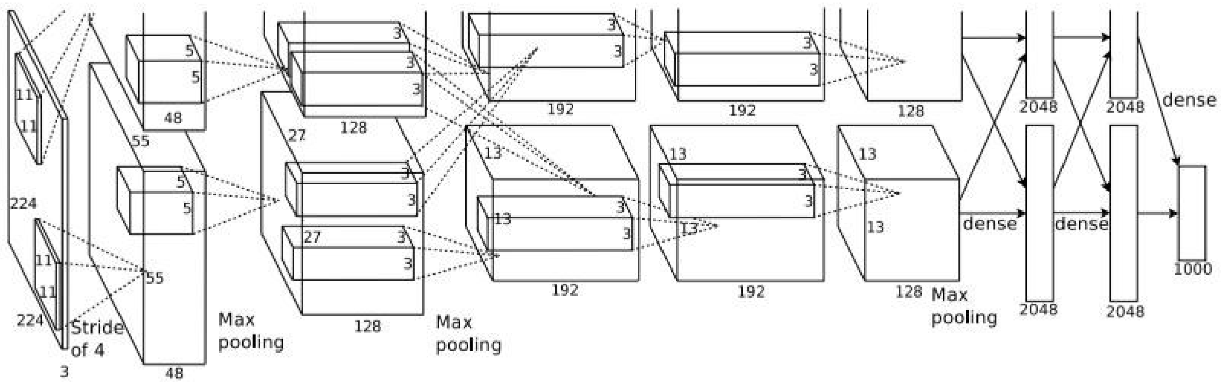


Figure 8-1 AlexNet Architecture

Once the model is defined, we will compile this model and use *Adam* as an optimizer. We train the model for 10 epochs.

On this classification problem, AlexNet's findings are not promising. We observed the validation accuracy 99 percent but it was fluctuating all the way. On the training set, the algorithm learns quickly, but it is unreliable on the validation set. It appears to be an example of overfitting, and the architecture is likely too sophisticated for this classification task. Also, we can see in the confusion matrix there is only one False-Positive and False-Negative value which is quite impressive.

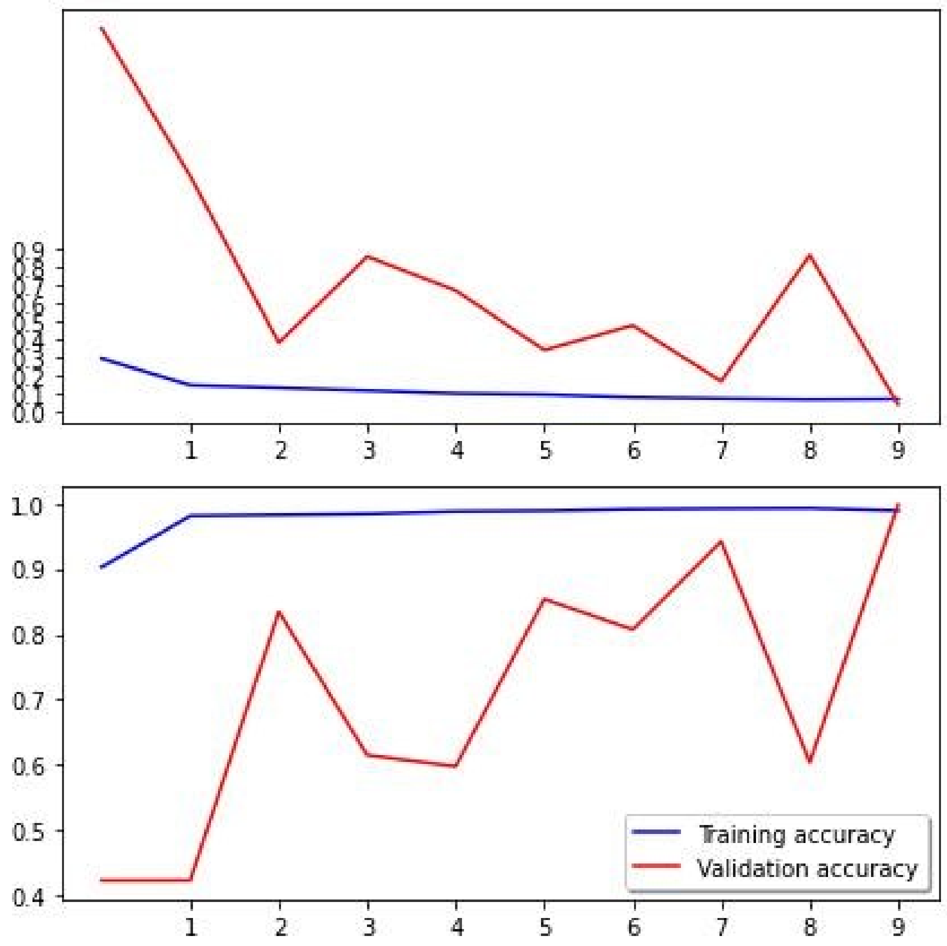


Figure 8-2 Accuracy and loss for training and validation dataset

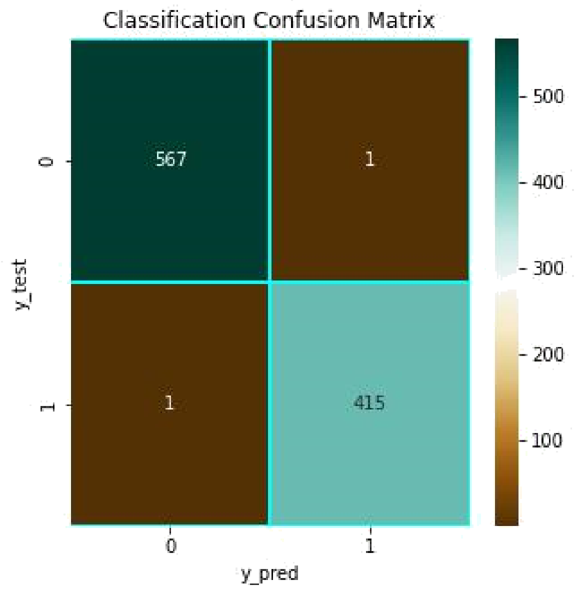


Figure 8-3 Confusion Matrix

9 LeNet-5 Architecture

Yann André LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, a French American computer scientist, created this very old neural network architecture in 1998. The architecture has seven layers, including two sets of Convolution layers and two sets of Average pooling layers, followed by a flattening convolution layer. Following that, there are two dense fully linked layers and a SoftMax classifier.

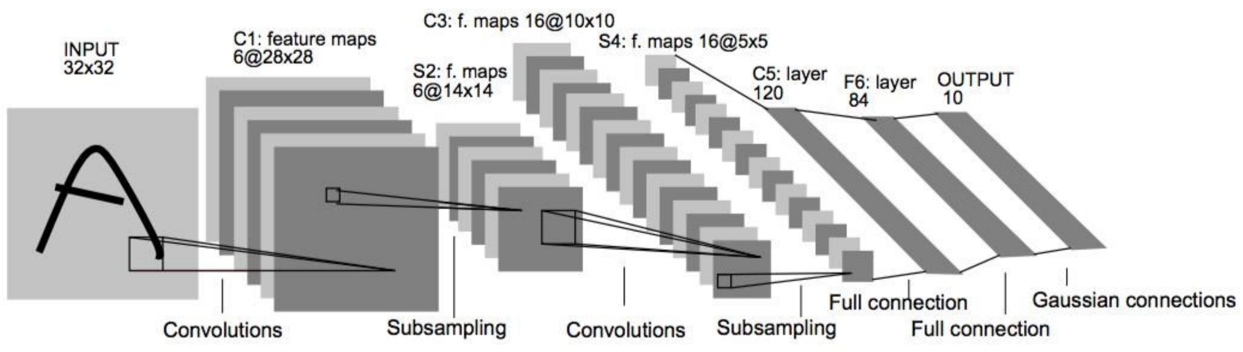


Figure 9-1: LeNet-5 Architecture

The activation function we use is ReLU in each layer except the output layer we use Sigmoid. The loss function we are using here is categorical_crossEntropy. It is used to determine how good the projected value is compared to the actual value. A loss function simply indicates how far the expected value differs from the actual value. Adam has been used as an optimization algorithm in conjunction with the loss function. We train our model for 10 epochs with a 10^{-6} Learning rate. As we can see in the graph, we got a validation accuracy of 97 percent with the least loss. LeNet provides enough performance and a steady solution that shows no signs of overfitting. Due to the problem's simplicity, the model learns quickly and produces a fairly accurate classification in the validation set as early as the first epoch.

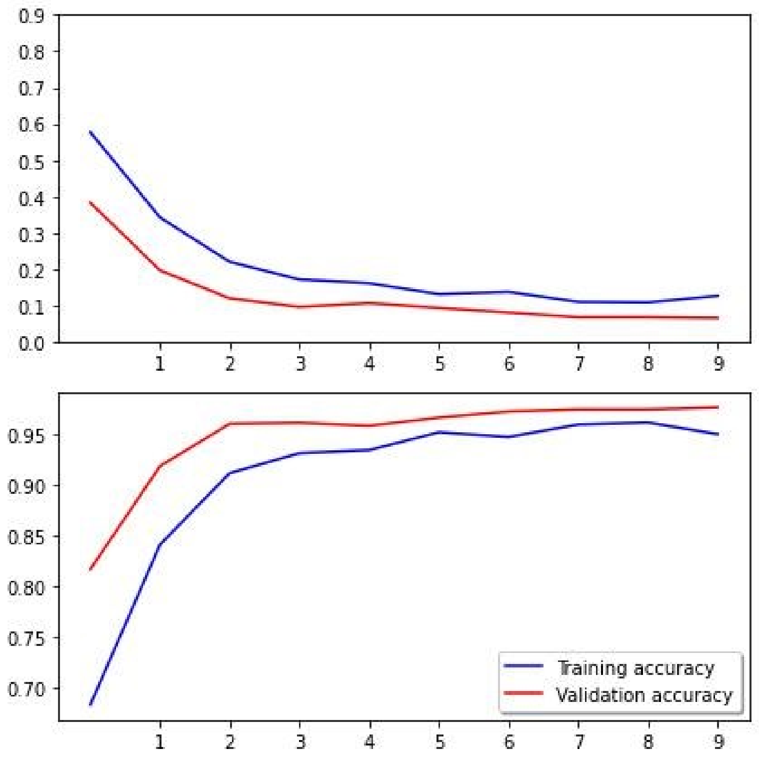


Figure 9-2 Accuracy and loss for training and validation dataset

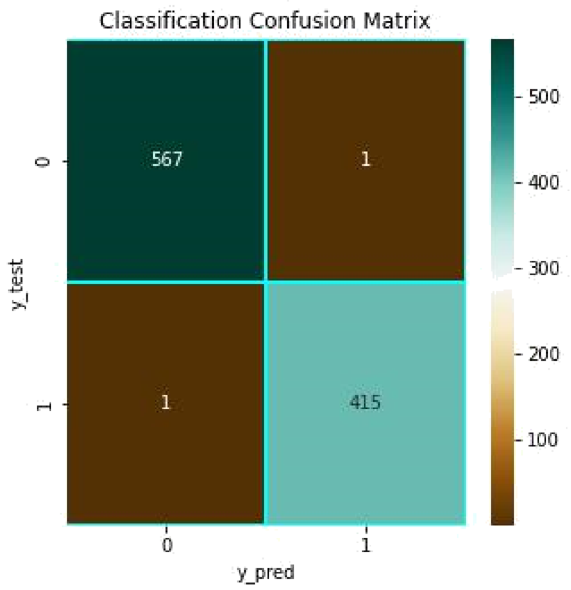


Figure 9-3 Confusion Matrix

10 Conclusion:

In the light of findings and observations of our research, we can conclude that:

- Higher complexity in the model improved the performance, but it is necessary to calibrate the complexity of the network to avoid overfitting/underfitting problems.
- We observed Cross-validation method is preferred over hold-out because it allows your model to train on multiple train-test splits and this strategy gave good results, but it is very expensive. On the other hand, hold-out is less expensive and reliant on a single train-test split so that the hold-out method's score is influenced by the way the data is divided into train and test sets.
- In mainstream architecture, we observed overall good performance of LeNet-5 architecture as compared to AlexNet.

11 Bibliography:

Freecodecamp.org

towardsdatascience.com

https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f#:~:text=Cross%2Dvalidation%20is%20usually%20the,just%20one%20train%2Dtest%20split

https://medium.com/@mgazar/lenet-5-in-9-lines-of-code-using-keras-ac99294c8086

https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342